An Examination of the Effects of Requirements Changes on Software Releases

George Stark, *IBM Global Services*
Al Skillicorn, *The MITRE Corporation*
1st Lt. Ryan Ameele, *U.S. Air Force*

Requirements are the foundation of the software release process. They provide the basis to develop budgets, schedules, and design and testing specifications. Changing requirements during a software release process impacts the cost, schedule, and quality of the product that results. We have collected data on 40 software releases in our environment to understand the source, magnitude, and effects of changing requirements on software maintenance releases. The benefits received include better management of releases and improved customer communications.

Several authors have noted that maintenance of software systems intended for a long operational life pose special management problems [1-3]. The Software Engineering Institute believes that organizational processes are a major factor in the predictability and quality of software [4]. J. Arthur and K. Stevens explain that descriptiveness, completeness, and readability of software documentation are key factors affecting system maintainability [5]. Additionally, M. Hariza, et al., B. Curtis, and C. Yuen all conclude that programmer experience is at least as important as code attributes in determining the complexity associated with software maintenance [3,6,7]. Research by the Standish Group and W. Wayt Gibbs indicates that a low software success rate results from poor requirements and poor risk management [8, 9]. Therefore, software maintenance planning and management should be formalized and quantified.

Requirements are the foundation of the software release process. They provide the basis to develop budgets, schedules, and design and testing specifications. In the maintenance environment, requirements are gathered through change requests from a variety of people including decision makers, system operators, developers, and external interface teams. These people have different backgrounds and different levels of understanding of computers and system operations. This diversity often leads to misinterpretation of the intent of the change description, which can change the scope of the requirement.

Furthermore, throughout the release process, requirements often change. During release planning, requirements analysis, design, and test reviews, new priorities are established, and changes to the release content are requested in the form of change requests being added or deleted from the release. This requirements volatility makes it difficult to develop dependable release schedules and budgets. B. Curtis, H. Krasner, and N. Iscoe conclude that accurate problem domain knowledge is critical to the success of a project, and requirements volatility causes major difficulties during development [10]. Although these conclusions confirm most people's intuitions concerning requirements volatility, they are not precise enough to help managers take effective action on their projects. M. Lubars, C. Potts, and C. Richter went further by interviewing 23 project teams and recommending organizational solutions rather than technological solutions to the requirements analysis issue [11]. In no case did they find a coherent relationship between requirements analysis and project planning.

This article therefore has two major goals: first, to present an organization's data regarding the source, timing, and impact of requirements volatility on the project planning process; and second, to describe opportunities for management action in the project planning process.

Organizational Data on Requirements Volatility and Project Planning

The Organization and the Data Collected

In 1994, the Missile Warning and Space Surveillance Sensors (MWSSS) Program Management Office was assigned responsibility for the maintenance of seven products executing in 10 locations worldwide. Combined, the products contained 8 million source lines of code written in 22 languages. Some of the systems were more than 30 years old, and the newest system became operational in 1992. They all operated in hard real-time environments and had a small set of users. To support the management of these products, we instituted the measurement program defined in [12].

In this project environment, a requirement was defined as an approved change request. The customer and supplier agreed to a set of requirements and a project plan to deliver a new version of a product. A requirements change was either an added change request, a deleted change request, or a change in scope to an agreed-on change request in the version content. Because requirements management was a primary factor in our success, we collected data on

- Type of requirement.
- Planned and actual effort days for each requirement.
- Planned and actual number of calendar days for a version.
- Requirements changes made to the version after plan approval—type of change, requesting group, and impact.

Who, How Often, and What Kind of Requirements Changes

To better understand our environment and how to improve it, we needed to answer the following questions.

- Who requests requirements changes?

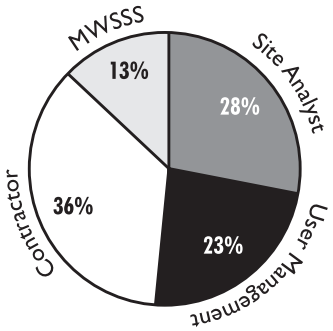


Figure 1. *Requirements changes by source.*

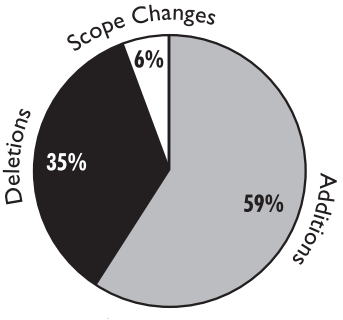


Figure 2. *Requirements changes by type.*

- How often do our releases experience requirements changes?
- What kind of changes are most common?
- How much effort is associated with individual requirements?

Four groups contributed to the release process: contractor development team, acquisition management team, user management, and site analysts. Each of these groups contributed to the requirements changes associated with a release. Figure 1 shows the percent of changes made by each group.

Requirements volatility comes in three types: additions to the delivery content, deletions from the delivery content, and changes in scope to an agreed-on requirement. A total of 108 requirements changes were made during 40 software releases since 1994. Figure 2 shows the distribution of these changes by type. Additions to the release content were the most common form of change, followed by deletions, with scope change being relatively rare.

Figure 3 shows the requirements volatility for each of the 40 deliveries. Fourteen of the 40 deliveries (35 percent) had no requirements change. Of the 14 deliveries, six were made on or ahead of schedule, four were within 15 percent of the original scheduled date, and four were more than 15 percent late. Twenty-six of the 40 (65 percent) had requirements change, with eight of them having greater than 50 percent change. Of the 26 releases that experienced requirements change, 16 had requirements added, 15 had deletions, and four had scope changes. Seven releases had a combination of adds, deletes, or changes.

To understand how much effort was associated with individual changes, we developed the software change taxonomy shown in Table 1. It includes 10 types of changes and root causes for each change type.

We categorized the changes delivered in eight releases using this taxonomy, which consisted of 104 modification changes (43 percent) and 139 fix changes (57 percent). Figure 4 is a Pareto diagram of this change data. The left vertical axis shows the number of changes attributed to each class, and the right vertical axis represents the cumulative percentage of defects and is a convenient scale from which to read the line graph. The line graph connects the cumulative percents (and counts) at each category.

Figure 3. *Requirements volatility for 40 deliveries.*

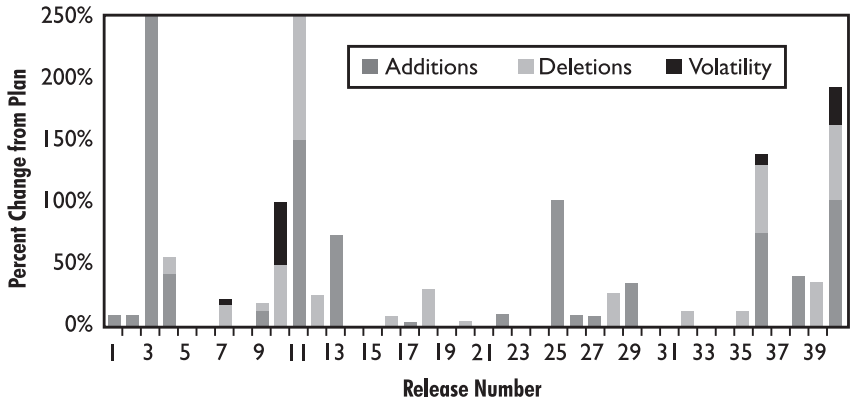


Table 1. *Software change taxonomy.*

Change Type	Root Cause
Computational	Incorrect operand in equation.
	Incorrect use of parentheses.
	Incorrect or inaccurate equation.
	Rounding or truncation error.
Logic	Incorrect operand in logical expression.
	Logic out of sequence.
	Wrong variable being checked.
	Missing logic or condition test.
	Loop iterated incorrect number of times.
Input	Incorrect format.
	Input read from incorrect location.
	End-of-file missing or encountered prematurely.
Data Handling	Data file not available.
	Data referenced out-of-bounds.
	Data initialization.
	Variable used as flag, or index not set properly.
	Data not properly defined or dimensioned.
	Subscripting error.
Output	Data written to different location.
	Incorrect format.
	Incomplete or missing output.
	Output garbled or misleading.
Interface	Software and hardware interface.
	Software and user interface.
	Software and database interface.
	Software and software interface.
Operations	COTS or GOTS software change.
	Configuration control.
Performance	Time limit exceeded.
	Storage limit exceeded.
	Code or design inefficient.
	Network efficiency.
Specification	System-to-system interface specification incorrect or inadequate.
	Functional specification incorrect or inadequate.
	User manual or training inadequate.
Improvement	Improve existing function.
	Improve interface.

Figure 4 indicates that logic changes to the software are most common (45 changes or 19 percent of the total). (Although not shown in Figure 4, the majority root cause is missing logic or condition tests for error handling.) Using this information, we have our design and code reviews to specifically look for these logic problems. Only two of the 243 changes involved data input problems.

Figure 5 is a Pareto diagram of the effort required to make each change. It shows that although changes based on specification changes only ranked fourth in number of changes with 26, they accounted for 20 percent of the total effort at 591 staff-days. Logic changes fall to sixth when viewed in this manner.

The information from this analysis helped maintenance engineers make better requirements cost estimates. By reviewing change requests and accurately assigning them to the change taxonomy, they could estimate the staff-days required to design, code, and test changes. For example, the average staff-days of effort required for changes to interface requirements are 24 staff-days with a standard deviation of 50 staff-days, whereas the average for functional specification changes is 23 staff-days with a standard deviation of 29 staff-days. Next, the actual was tracked against the estimate, and the taxonomy and cost information was updated as each release was completed.

Although the current information is highly variable for each root cause, the effort data is expected to converge around a reasonable mean as more data is collected. This will increase our confidence in the estimates. Sudden changes could indicate a need to re-examine our processes or a need to change the staff that implements the requirement. Even with the current variability, using historical data is the best method to estimate individual change effort.

A Microview of Requirements Changes on One Release

The Configuration Control Board approved a release plan to deliver 17 requirements in nine months at a cost of approximately $490,000. Figure 6 shows the requirements changes over time for this release. These changes were processed both formally (through the Configuration Control Board) and informally (agreement between users and developers). The figure also shows that a total of 20 changes were made to the release content in the 14 months since project plan approval. The two spikes for February and October occurred after design reviews where major scope changes occurred with some of the requirements. Nine of the changes occurred in the last five months of the effort, and only six of the delivered requirements were a part of the original approved plan. This greatly impacted the implementation effort.

Requirements Changes by Type

Figure 7 shows a significantly different distribution of changes by type than the overall distribution of Figure 3. In Figure 7, scope changes account for 26 percent of the changes to the release compared with 8 percent for all releases. The increase in the amount of scope changes was a major factor in the delivery schedule, which illustrates two important points: general distribution should only be used as a planning guide, and releases should be managed as stand-alone projects.

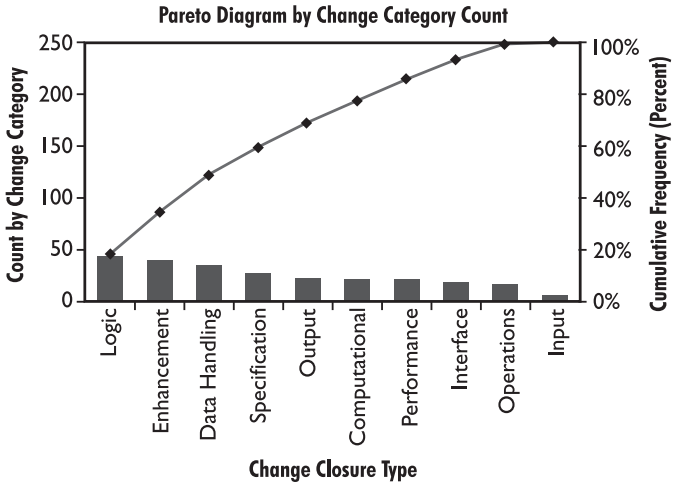


Figure 4. *Software maintenance changes by type.*

Requirements Changes by Source

Requirements changes could be initiated by the customer (analysts or management personnel) or the development team, i.e., the contractor or the MWSSS Program Management Office. Figure 8 shows the distribution of changes by source for this release. This chart shows that the changes were distributed as 55 percent driven by the development team and 45 percent by the customer. The analyst personnel and the devel-

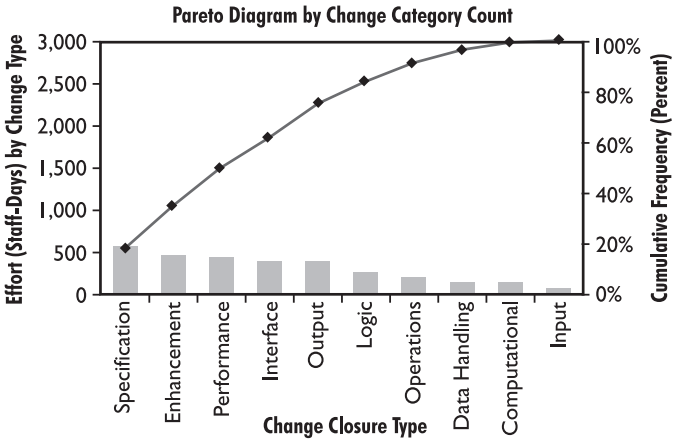


Figure 5. *Staff-days of effort by category.*

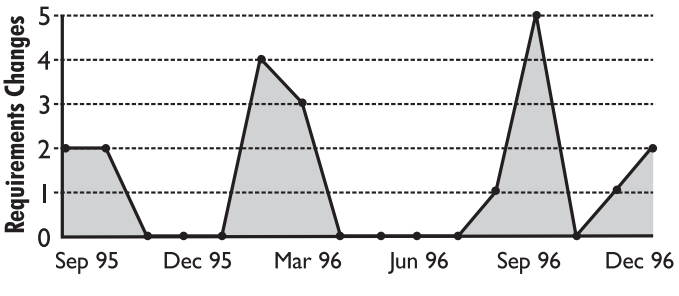


Figure 6. *Requirements changes by month for one release.*

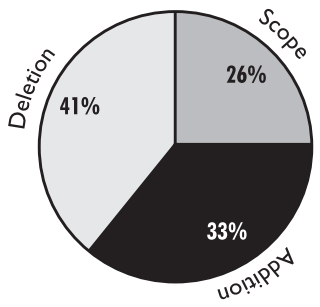


Figure 7. *Requirements changes by type.*

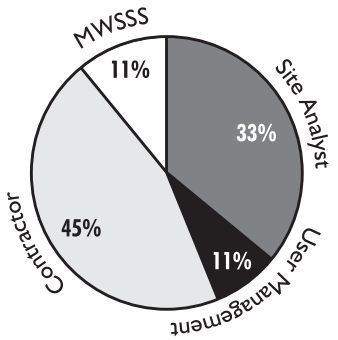


Figure 8. *Requirements changes by source.*

opment contractor accounted for 80 percent of the changes (16 out of 20).

Schedule, Cost, and Quality Impact of Changes

Figure 9 shows the predicted version operational date over time for the project. The first slip (three and one-half months) was reported at the design review held three months after project start. A second slip (three weeks) was announced eight months into the project. Finally, another completion date, this one four and one-half months later, was announced one year after project start. These announced schedule slips correspond to the major jumps in the requirements changes graph (Figure 6). Two defects that required rework and more testing were reported during operational testing of this release. The release was delivered a month later, making the total schedule 10 months (more than double the original project plan) and the cost $100,000 (22 percent over budget). Of course, requirement volatility was not the only reason for the schedule and cost overrun, but it was the major factor.

Observations and Recommendations

Requirements must be more clearly explained and understood by the development team, and change agreements must be more formally managed by the management team responsible for the software releases. Accordingly, we changed our process to include a rigorous requirements review meeting with the customer prior to presenting the release plan for Configuration Control Board approval. We also have biweekly meetings with the MWSSS management where the project requirements status and other project issues are briefed.

A Macromodel to Forecast the Effects of Requirements Changes on Releases

To help release teams better manage the requirements volatility and get a handle on the impact of changes to their project, we began to develop models based on the historical release data. Table 2 shows the percent of planned schedule achieved (100 means the plan was met, greater than 100 means late, less than 100 means delivered early), the square root of the percent of requirements volatility (the sum of all changes), and the productivity risk associated with the 20 releases. The square root transformation was used to spread out the numbers close to zero and condense the numbers greater than one. Risk is defined as changes closed per effort days available.

Figures 10 and 11 are scatter plots of the percent of planned schedule vs. the other two descriptive variables in Table 2. Individually, these plots have little correlation, but used together, these variables can provide insight to project managers to help them understand the schedule impact of requirements changes.

A linear regression analysis was performed on the data to develop a model to predict schedule impact based on requirements volatility and risk with the following results:

$$Y = 0.97 + 0.41*X^{1/2} + 0.23*Z \quad (1)$$

where

Y = Percent Schedule Change
X = Requirements Volatility
Z = Risk

The proportion of variance explained by this model (R^2) is 0.72, and the standard error of the estimate is 0.17. Notice that the schedule change goes up regardless of whether the requirements changes were an addition or deletion because the input to the model is percent of requirements changes. This is a topic of debate in the organization: Some argue that removing requirements involves effort to change the design and test procedures, whereas others argue that a reduction in requirements means less work for the team and earlier completion of the project.

Figure 12 shows the results of applying the model to all 40 releases executed by our organization. From this figure, it can be seen that the model performs much better in the 115 percent to 130 percent of planned schedule range and yields more optimistic results as predictions get larger, i.e., greater than 150 percent of plan. This may indicate the

Table 2. *Schedule, requirements volatility, and risk data for 20 software maintenance versions.*

Version Content	Percent of Planned Schedule	SQRT (Percent of Requirements Change)	Risk (Change Requests per Staff-Day)
1	108	33	0.14
2	104	32	0.15
3	168	158	0.50
4	132	76	0.18
5	115	0	0.08
6	115	48	0.27
7	118	45	0.16
8	139	100	0.01
9	219	158	0.19
10	129	50	0.07
11	100	87	0.07
12	111	0	0.01
13	102	18	0.12
14	123	55	0.07
15	92	0	0.20
16	178	245	0.01
17	104	0	0.02
18	110	23	0.08
19	100	31	0.12
20	94	0	0.03

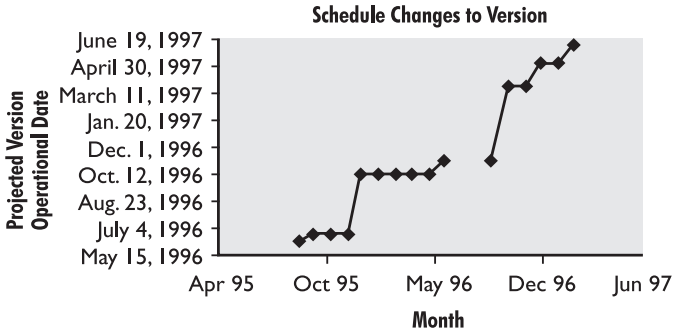


Figure 9. *Predicted version operational date by month.*

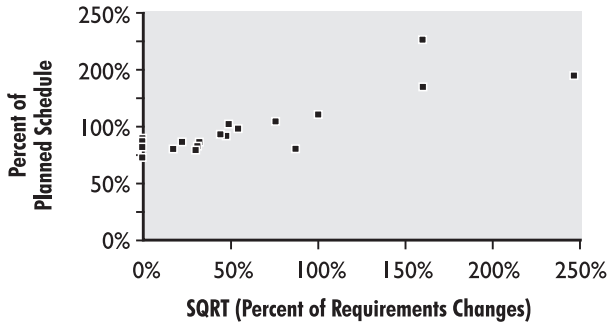


Figure 10. *Percent of planned schedule vs. square root (requirements volatility) for 20 versions.*

need for another explanatory variable as major changes occur to releases.

Macromodel Use and Benefits

We have used this equation to explain the expected impact of changes to the delivery plan as they arise. For example, one version contained 15 planned requirements scheduled for delivery in 91 calendar days—the customer wanted to drop two of the requirements and change the scope of a third at preliminary design. Managers estimated the risk to version delivery to change from 0.14 (15 changes in 108 staff-days) to 0.1 (13 changes in 130 staff-days). Using the model, managers forecasted the overall schedule impact to be [0.97 + $0.41*(0.2)^{1/2}$ + 0.23*(0.1)] = 1.18 or an 18 percent schedule slip. An 18 percent slip is equivalent to 16 days added to the 91-day schedule. These 16 days would have cost the customer an additional $60,000.

During discussion about the model and the prediction, the customer decided that this schedule slip was not acceptable to the overall mission of the version; therefore, they decided not to pursue the changes but to incorporate the scope change in the next release. The metrics-based model facilitated objective communication with the customer concerning version release plans and status.

The model forecasted a $50,000 cost impact and a 12-day schedule slip from a second customer request to change the release content. The additional cost was not acceptable to the customer, so they decided to incorporate the changes in the next release. Thus, the overall cost avoidance because of quantitative schedule impact analysis was $110,000.

Conclusion

Requirements management involves establishing and maintaining an agreement between the customer and the supplier on the specific number and technical content of the performance and functionality that will be included in a software release. This agreement forms the basis to estimate, plan, perform, and track the project's activities. We believe other organizations can benefit from our experience.

Acknowledgments

We thank Dieter Rombach for his suggestions and for providing references for this article. We also thank the many referees for their excellent reviews. ◆

About the Authors

George Stark is a programming consultant with the IBM Corporation in Austin, Texas. Previously, he was a principal scientist with The MITRE Corporation, where he supported the software efforts of the MWSSS Program Management Office. His technical interests include software metrics and reliability for management decision making. He has been involved in software reliability measurement for 15 years and was the vice chairman of the American Institute of Aeronautics and Astronautics blue-ribbon panel on software reliability. He has been the manager of software testing and reliability for a local loop fiber-optic telephone system. He received the Johnson Space Center Quality Partnership Award and the MITRE General Manager's Award for contributions to software measurement. He has a bachelor's degree in statistics from Colorado State University and a master's degree in mathematics from the University of Houston.

Figure 11. *Percent of planned schedule vs. delivery risk.*

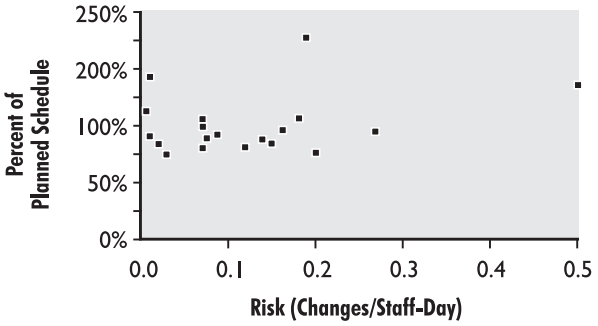
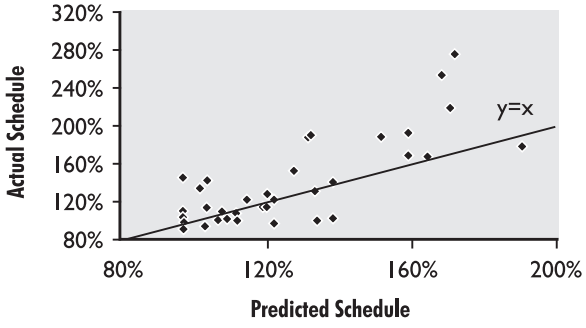


Figure 12. *Actual vs. predicted schedule using linear model for all 40 releases.*



IBM Global Services
11400 Burnet Road, MD 3901
Austin, TX 78759
Voice: 512-823-8515
Fax: 512-823-3385
E-Mail: gstark@us.ibm.com

Al Skillicorn is a member of the technical staff of The MITRE Corporation. He supports the software maintenance of the early warning radar systems. Among his other responsibilities are the Year 2000 problem and future software architectures. He has a bachelor's degree in engineering from the U.S. Military Academy at West Point. Previous work included communications modeling and analysis for the Regency Net Communication System and for the Theatre Nuclear Forces Communications System in Europe.

The MITRE Corporation
1150 Academy Park Loop #212
Colorado Springs, CO 80910
Voice: 719-556-2565
E-mail: skilliad@cisf.af.mil

1st Lt. Ryan Ameele is the software process manager for the MWSSS Program Management Office. Previously, he was the Cargo System Software Development Team leader for the Air Mobility Command Computer System Squadron at Scott Air Force Base, Ill. He has a bachelor's degree in engineering from Clarkson University in New York. He was recently selected for promotion to captain.

SSSG/SDWSE
1050 E. Stewart Avenue
Peterson AFB, CO 80914-2902
Voice: 719-556-9906
E-mail: ameeler1@cisf.af.mil

References

1. Card, D.N., D.V. Cotnoir, and C.E. Goorevich, "Managing SW Maintenance Cost and Quality," *Proceedings of the International Conference on Software Maintenance*, September 1987.
2. Chapin, N., "The Software Maintenance Life-Cycle," *Proceedings of the International Conference on Software Maintenance*, 1988.
3. Hariza, M., J.F. Voidrot, E. Minor, L. Pofelski, and S. Blazy, "Software Maintenance: An Analysis of Industrial Needs and Constraints," *Proceedings of the International Conference on Software Maintenance*, Orlando, Fla., 1992.
4. Software Engineering Institute, "Software Process Maturity Questionnaire Capability Maturity Model, Version 1.1," Carnegie Mellon University, Pittsburgh, Pa., 1994.
5. Arthur, J. and K. Stevens, "Assessing the Adequacy of Documentation Through Document Quality Indicators," *Proceedings of the International Conference on Software Maintenance*, 1989.
6. Curtis, B., "Conceptual Issues in Software Metrics," *Proceedings of the IEEE International Conference on System Sciences*, 1986.
7. Yuen, C., "An empirical Approach to the Study of Errors in Large Software Under Maintenance," *Proceedings of the International Conference on Software Maintenance*, 1985, pp. 96-105.
8. The Standish Group, "The Scope of Software Development Project Failures," Dennis, Mass., 1995.
9. Gibbs, W. Wayt, "Software's Chronic Crisis," *Scientific American*, September 1994, pp. 72-81.
10. Curtis, B., H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM*, Vol. 31, No. 11, 1988, pp. 1268-1287.
11. Lubars, M., C. Potts, and C. Richter, "A Review of the Practice in Requirements Modeling," *Proceedings of the International Symposium on Requirements Engineering*, 1996, pp. 2-14.
12. Stark, G.E., "Measurements for Managing Software Maintenance," *Proceedings of the International Conference on Software Maintenance*, Monterey, Calif., November 1996, pp. 152-161.